


```
PPPPPPPP  UU      UU  TTTTTTTTTT  CCCCCCCC  AAAAAA  CCCCCCCC  HH      HH  EEEEEEEEEE
PPPPPPPP  UU      UU  TTTTTTTTTT  CCCCCCCC  AAAAAA  CCCCCCCC  HH      HH  EEEEEEEEEE
PP      PP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PP      PP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PP      PP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PPPPPPPP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PPPPPPPP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PP      PP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PP      PP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PP      PP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
PP      PP  UU      UU      TT      CC      AA      AA      CC      HH      HH  EEEEEEEEEE
UUUUUUUUUU  TT      CCCCCCCC  CCCCCCCC  AA      AA      CCCCCCCC  HH      HH  EEEEEEEEEE
UUUUUUUUUU  TT      CCCCCCCC  CCCCCCCC  AA      AA      CCCCCCCC  HH      HH  EEEEEEEEEE

LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```



```
1 0001 0 MODULE LBR_PUTCACHE ( ! General library procedure routines
2 0002 0 LANGUAGE (BLISS32),
3 0003 0 IDENT = 'V04-000'
4 0004 0 ) =
5 0005 1 BEGIN
6 0006 1
7 0007 1
8 0008 1 *****
9 0009 1 *
10 0010 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
11 0011 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
12 0012 1 * ALL RIGHTS RESERVED.
13 0013 1 *
14 0014 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
15 0015 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
16 0016 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
17 0017 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
18 0018 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
19 0019 1 * TRANSFERRED.
20 0020 1 *
21 0021 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
22 0022 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
23 0023 1 * CORPORATION.
24 0024 1 *
25 0025 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
26 0026 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
27 0027 1 *
28 0028 1 *
29 0029 1 *****
30 0030 1
31 0031 1 ++
32 0032 1
33 0033 1 FACILITY: Library access procedures
34 0034 1
35 0035 1 ABSTRACT:
36 0036 1
37 0037 1 The VAX/VMS librarian procedures implement a standard access method
38 0038 1 to libraries through a shared, common procedure set.
39 0039 1
40 0040 1 ENVIRONMENT:
41 0041 1
42 0042 1 VAX native, user mode.
43 0043 1
44 0044 1 --
45 0045 1
46 0046 1
47 0047 1 AUTHOR: Bob Grosso 23-Jan-1981
48 0048 1
49 0049 1 MODIFIED BY:
50 0050 1
51 0051 1 V03-001 JWT0067 Jim Teague 24-Nov-1982
52 0052 1 Make LBR clean up after itself more thoroughly.
53 0053 1
54 0054 1 --
```



```
: 56      0055 1 LIBRARY
: 57      0056 1      'SYSS$LIBRARY:STARLET.L32';      ! System macros
: 58      0057 1 REQUIRE
: 59      0058 1      'PREFIX';      ! Librarian general definitions
: 60      0197 1 REQUIRE
: 61      0198 1      'LBRDEF';      ! Librarian structure definitions
: 62      0789 1
: 63      0790 1 EXTERNAL LITERAL
: 64      0791 1      lbr$_badparam,
: 65      0792 1      lbr$_writeerr,
: 66      0793 1      lbr$_normal;
: 67      0794 1
: 68      0795 1 EXTERNAL
: 69      0796 1      lbr$_gl_rmsstv,      ! Returns STV on RMS errors
: 70      0797 1      lbr$_gl_control: REF BBLOCK; ! Pointer to current control block
: 71      0798 1
: 72      0799 1 EXTERNAL ROUTINE
: 73      0800 1      check_lock : JSB 0,      ! Check if index is locked.
: 74      0801 1      dealloc_mem : JSB 2,      ! To return cache.
: 75      0802 1      get_mem : JSB 2,      ! To allocate empty_cache_buffer
: 76      0803 1      empty_cache,      ! To empty cache when library read only
: 77      0804 1      write_block : JSB 2,      ! To write out block of highest VBN
: 78      0805 1      lookup_cache : JSB 2,      ! To find block of highest VBN
: 79      0806 1      validate_ctl : JSB 1;      ! To validate the control index for lbr$_flush
: 80      0807 1
: 81      0808 1 FORWARD ROUTINE
: 82      0809 1      write_n_blocks,      ! Write empty cache buffer
: 83      0810 1      flush_cache,      ! Write out and deallocate cache
: 84      0811 1      dealloc_cache,      ! Deallocate hash table and cache
: 85      0812 1      test_write;      ! Write out highest VBN in cache to ensure
: 86      0813 1      ! sufficient disk space.
: 87      0814 1
```



```

89 0815 1 ROUTINE flush_cache ( flush_data, writestatus, continue_on_err ) =
90 0816 1 |+++
91 0817 1 |
92 0818 1 | This routine writes out to the library either all the data blocks
93 0819 1 | or all the index blocks. The emptying of the cache starts with
94 0820 1 | the first bucket and proceeds by emptying a bucket at a time. If
95 0821 1 | a block has been modified then those VBN's in the cache which are
96 0822 1 | sequentially adjacent to it are also copied to the empty_cache buffer.
97 0823 1 | The next VBN in sequential order will be found in the next bucket, so
98 0824 1 | the search will continue along the buckets until the buffer fills or
99 0825 1 | the next VBN is not found in the next bucket, at which point the
100 0826 1 | buffer will be written out to the library.
101 0827 1 | Before any blocks are written, the block with the largest VBN is
102 0828 1 | written out to ensure that there is room on the storage device to
103 0829 1 | update the library.
104 0830 1 |
105 0831 1 | Inputs
106 0832 1 |
107 0833 1 | flush_data = lbr$c_flushdata if data blocks are to be flushed,
108 0834 1 | lbr$c_flushall if index blocks are to be flushed.
109 0835 1 |
110 0836 1 | writestatus = If writestatus indicates an error then no more
111 0837 1 | blocks will be written. If writestatus = true,
112 0838 1 | then the cache will continue to be deallocated.
113 0839 1 |
114 0840 1 | continue_on_err =
115 0841 1 | If true, then continue to deallocate cache after
116 0842 1 | a buffer write error has occurred, else exit.
117 0843 1 |
118 0844 1 | Routine value
119 0845 1 |
120 0846 1 | success or status from write.
121 0847 1 | ---
122 0848 2 BEGIN
123 0849 2 |
124 0850 2 |+++
125 0851 2 | body of ROUTINE flush_cache
126 0852 2 | ---
127 0853 2 |
128 0854 2 BIND
129 0855 2 context = .lbr$gl_control [lbr$l_ctxptr] : BBLOCK,
130 0856 2 cache = .context [ctx$l_cache] : VECTOR;
131 0857 2 |
132 0858 2 LITERAL
133 0859 2 first_bucket = 0, !Offset of first entry in hash table
134 0860 2 last_bucket = lbr$c_hashsize/4 - 1; !Offset of last entry in hash table
135 0861 2 |
136 0862 2 LOCAL
137 0863 2 flushbuf : BBLOCK [lbr$c_flshbfsiz * lbr$c_pagesize],
138 0864 2 | buffer when called by lbr$flush
139 0865 2 putcachebuf : REF BBLOCK, | point to either flushbuf which is on stack
140 0866 2 | or buffer allocated from virtual memory.
141 0867 2 putbufsiz, | Size of the buffer being used
142 0868 2 bufstatus, | allocation status for vm buffer
143 0869 2 status;
144 0870 2
145 0871 2
```



```
146 0872 2 !+++
147 0873 2 Descend through hash table a bucket at a time.
148 0874 2 For each entry in a bucket, check it's vbn and then search the next bucket
149 0875 2 for the next vbn in sequence.
150 0876 2 If found, store in buffer so all consecutive vbn's can be written with one QIO.
151 0877 2 Keep getting consecutive vbn's until buffer full or next in sequence not found.
152 0878 2 A count is kept of the number of virtual blocks copied to the buffer, and
153 0879 2 the location in the buffer of the last vb that was modified is recorded so
154 0880 2 that trailing unmodified blocks are not written out.
155 0881 2 ---
156 0882 2
157 0883 2 If a buffer cannot be allocated from virtual memory then use the buffer
158 0884 2 on the stack.
159 0885 2
160 0886 2 putbufsiz = lbr$_putbufsiz;
161 0887 2 IF NOT (bufstatus = get_mem (lbr$_putbufsiz * lbr$_pagesize, putcachebuf))
162 0888 2 THEN
163 0889 2 BEGIN
164 0890 2 putbufsiz = lbr$_flshbufsiz;
165 0891 2 putcachebuf = flushbuf;
166 0892 2 END;
167 0893 2 INCR bucket FROM first_bucket TO last_bucket DO ! for every bucket in hash table
168 0894 2 BEGIN
169 0895 2 LOCAL
170 0896 2 entry : REF BBLOCK,      ! Each bucket contains the head of a linked
171 0897 2                          ! list of cache entries.
172 0898 2 last_entry : REF BBLOCK,  ! Backward link in bucket's linked list
173 0899 2 nextcache_entry,         ! Next cache entry in the linked list.
174 0900 2 targvbn,               ! The virtual block number of the first block
175 0901 2                          ! in the empty cache buffer.
176 0902 2 bufblkcnt,             ! Count of the blocks written into the
177 0903 2                          ! putcachebuf buffer.
178 0904 2 lstmodblkloc,          ! The count of the last vbn in the putcachebuf
179 0905 2                          ! which had the modified bit set. The buffer
180 0906 2                          ! contents will be written out up to the
181 0907 2                          ! last modified block.
182 0908 2 next_vbn,              ! The next vbn in sequence after the one just
183 0909 2                          ! copied to the putcachebuf.
184 0910 2 next_bucket;          ! Record location of the hash table bucket
185 0911 2                          ! being searched.
186 0912 2
187 0913 2 last_entry = cache[.bucket] ;      ! back link in hash bucket linked list
188 0914 2 entry = .cache [.bucket];        ! first hash list entry
189 0915 2
190 0916 2 Loop for each entry in the bucket unless there has been a write error
191 0917 2 and continue_on_err is false
192 0918 2
193 0919 2 WHILE ((.entry NEQ 0) AND (.writestatus OR .continue_on_err) )DO
194 0920 2 BEGIN
195 0921 2 nextcache_entry = .entry [cache$_link]; ! remember the next entry in bucket
196 0922 2 IF .entry [cache$_v_data] EQL .flush_data
197 0923 2 THEN
198 0924 2 BEGIN
199 0925 2 IF .entry [cache$_v_dirty] ! if the page has been modified
200 0926 2 THEN
201 0927 2 BEGIN
202 0928 2 IF .writestatus
```



```
THEN
    As long as there haven't been any write errors then keep
    copying blocks to buffer and writing buffer to library.
BEGIN
    targvbn = .entry [cache$l_vbn]; ! vbn of first block in empty cache buffer.
    CH$MOVE (lbr$c_pagesize, CH$PTR(.entry[cache$l_address]),
            CH$PTR(.putcachebuf));
    bufblkcnt = 1; ! record the number of pages and modified
    lstmodblkloc = 1; ! pages copied to the buffer
    next_vbn = .entry[cache$l_vbn] + 1; ! the virtual block number of the page
    ! following the last copied to buffer

    if the next vbn is in the cache it will be in one of
    the entries in the next bucket. Check for end of hash
    table and loop around.
IF .bucket LSS last_bucket
THEN next_bucket = .bucket + 1
ELSE next_bucket = 0;

    Keep looping until the buffer is filled or the
    consecutive VBN is not found.
WHILE ( (.bufblkcnt LSS .putbufsiz) AND (.next_vbn NEQ 0) ) DO
    BEGIN
        LOCAL
            nxtbucket_entry : REF BBLOCK, ! forward link in chain of cache ent
            last_bucket_entry : REF BBLOCK, ! rear link
            nxtvbn_entry : REF BBLOCK; ! the cache entry of the vbn being sought.
            ! retain forward link.
            ! retain backward link.
            ! mark the next sequential
            ! vbn as unfound.
            !
            While there are entries in the bucket and the VBN being sought
            has not been found and the search has not gone beyond where the vbn
            would be, keep looping through the entries in the bucket.
            Also give up search if block was found but is of wrong type; data/index.
        WHILE ((.nxtbucket_entry NEQ 0) AND (.nxtvbn_entry EQL 0)) DO
            BEGIN
                IF .nxtbucket_entry[cache$l_vbn] GTR .next_vbn
                THEN
                    BEGIN
                        last_bucket_entry = .nxtbucket_entry;
                        nxtbucket_entry = .nxtbucket_entry[cache$l_link];
                    END
                ELSE
                    BEGIN
                        IF .nxtbucket_entry[cache$l_vbn] EQL .next_vbn
                        THEN
                            BEGIN
                                IF .nxtbucket_entry[cache$v_data] EQL .flush_data
                                THEN
```

```
203 0929 6
204 0930 6
205 0931 6
206 0932 6
207 0933 6
208 0934 7
209 0935 7
210 0936 7
211 0937 7
212 0938 7
213 0939 7
214 0940 7
215 0941 7
216 0942 7
217 0943 7
218 0944 7
219 0945 7
220 0946 7
221 0947 7
222 0948 7
223 0949 7
224 0950 7
225 0951 7
226 0952 7
227 0953 7
228 0954 7
229 0955 7
230 0956 8
231 0957 8
232 0958 8
233 0959 8
234 0960 8
235 0961 8
236 0962 8
237 0963 8
238 0964 8
239 0965 8
240 0966 8
241 0967 8
242 0968 8
243 0969 8
244 0970 8
245 0971 8
246 0972 9
247 0973 9
248 0974 9
249 0975 10
250 0976 10
251 0977 10
252 0978 10
253 0979 9
254 0980 10
255 0981 10
256 0982 10
257 0983 11
258 0984 11
259 0985 11
```


260	0986	12
261	0987	12
262	0988	12
263	0989	12
264	0990	12
265	0991	12
266	0992	12
267	0993	11
268	0994	11
269	0995	10
270	0996	10
271	0997	9
272	0998	8
273	0999	8
274	1000	8
275	1001	8
276	1002	8
277	1003	8
278	1004	8
279	1005	8
280	1006	8
281	1007	8
282	1008	9
283	1009	10
284	1010	10
285	1011	9
286	1012	10
287	1013	10
288	1014	10
289	1015	10
290	1016	10
291	1017	10
292	1018	10
293	1019	10
294	1020	10
295	1021	10
296	1022	10
297	1023	10
298	1024	10
299	1025	10
300	1026	10
301	1027	10
302	1028	10
303	1029	9
304	1030	9
305	1031	9
306	1032	9
307	1033	9
308	1034	9
309	1035	9
310	1036	9
311	1037	9
312	1038	9
313	1039	9
314	1040	8
315	1041	7
316	1042	7

```
BEGIN
    Next VBN found so unlink and mark it found
    last_bucket_entry[cache$L_link] = .nxtbucket_entry[cache$L_link];
    nxtvbn_entry = .nxtbucket_entry; ! record the unlinked cache entry
    ! which contains the VBN searched f
END
ELSE nxtbucket_entry = 0 ! Was found but was wrong type.
END
ELSE ! the next VBN is not in cache
    nxtbucket_entry = 0; ! End search
END;
END; ! WHILE searching for next vbn

IF .nxtvbn_entry EQL 0 ! the next VBN was not found
THEN next_vbn = 0
ELSE
    The next VBN was found, copy it into buffer if it was
    modified. If not modified, copy it in unless it would
    be the last in the buffer.
    BEGIN
    IF ( (.nxtvbn_entry[cache$V_dirty]) OR
        (.bufblkcnt + 1 NEQ .putbufsiz) )
    THEN
        BEGIN
        CH$MOVE (lbr$C_pagesize,
            CH$PTR (.nxtvbn_entry[cache$L_address]),
            CH$PTR ( .putcachebuf + .bufblkcnt*lbr$C_pagesize ) );
        bufblkcnt = .bufblkcnt + 1;
        IF .nxtvbn_entry[cache$V_dirty] THEN lstmodblkloc = .bufblkcnt;
        When setting next bucket for continued search of sequential
        vbn's check to see if it should loop around end of hash table.
        IF .next_bucket LSS last_bucket
        THEN next_bucket = .next_bucket + 1
        ELSE next_bucket = 0;
        next_vbn = .next_vbn + 1;
        END
    ELSE
        page was not modified and would be last in buffer so don't
        bother with it.
        next_vbn = 0;
        deallocate block and then the hash bucket entry
        dealloc_mem (lbr$C_pagesize, .nxtvbn_entry[cache$L_address]);
        dealloc_mem (cache$C_length, .nxtvbn_entry);
        END; ! else the vbn was found
    END; ! WHILE still filling buffer and finding the next VBN to go in it.
writestatus = write_n_blocks ( .putcachebuf, .targvbn, .lstmodblkloc );
```



```

317 1043 7
318 1044 7
319 1045 7
320 1046 7
321 1047 7
322 1048 7
323 1049 7
324 1050 8
325 1051 8
326 1052 8
327 1053 8
328 1054 8
329 1055 7
330 1056 6
331 1057 5
332 1058 5
333 1059 5
334 1060 5
335 1061 5
336 1062 5
337 1063 5
338 1064 5
339 1065 4
340 1066 4
341 1067 4
342 1068 4
343 1069 4
344 1070 2
345 1071 2
346 1072 2
347 1073 2
348 1074 1

      ! If there has been a write error then exit if flush_cache
      ! was called to flush the cache, else it was called to deallocate
      ! the cache, so keep on deallocating without writing.
      IF NOT .writestatus AND NOT .continue_on_err
      THEN
      BEGIN
      ! ** Note ** those blocks in buffer are lost
      IF .bufstatus
      THEN dealloc_mem (.putbufsiz * lbr$c_pagesize, .putcachebuf);
      RETURN .writestatus;
      END;
      END;
      ! IF writestatus (if only deallocating and not writing)
      END;
      ! If the first was modified
      last_entry[cache$l_link] = .entry[cache$l_link]; ! unlink
      ! deallocate block and then the hash bucket entry
      dealloc_mem (lbr$c_pagesize, .entry[cache$l_address]);
      dealloc_mem (cache$c_length, .entry);
      END
      ! If the block type was correct
      ELSE
      ! the block type was incorrect so hop over it.
      last_entry = .entry;
      entry = .nxtcache_entry;
      END
      ! WHILE covering all entries in the bucket
      ! DO the whole hash table
      END;
      IF .bufstatus THEN dealloc_mem (.putbufsiz * lbr$c_pagesize, .putcachebuf);
      RETURN .writestatus;
      END;
      ! flush_cache
```

```
.TITLE LBR_PUTCACHE
.IDENT \V04-000\
```

```
.EXTRN LBR$_BADPARAM, LBR$_WRITEERR
.EXTRN LBR$_NORMAL, LBR$_GL_RMSSTV
.EXTRN LBR$_GL_CONTROL, CHECK_LOCK
.EXTRN DEALLOC_MEM, GET_MEM
.EXTRN EMPTY_CACHE, WRITE_BLOCK
.EXTRN LOOKUP_CACHE, VALIDATE_CTL
```

```
.PSECT $CODE$,NOWRT,2
```

```
OFFC 00000 FLUSH_CACHE:
```

```

04 5E FDE0 CE 9E 00002
50 0000G CF D0 00007
5A 0E A0 D0 0000C
AE 1E D0 00010
51 1C AE 9E 00014
50 3C00 8F 3C 00018
0000G 30 0001D
6E 50 D0 00020
09 6E E8 00023
```

```

.WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11
MOVAB -544(SP), SP
MOVL LBR$_GL_CONTROL, R0
MOVL 14(R0), R10
MOVL #30, PUTBUFSIZ
MOVAB PUTCACHEBUF, R1
MOVZWL #15360, R0
BSBW GET_MEM
MOVL R0, _BUFSTATUS
BLBS BUFSTATUS, 1$
```

```

: 0815
: 0855
: 0886
: 0887
:
```


04	AC	OC	AB	04	AE	01	D0	00026	MOVL	#1, PUTBUFSIZ	0890
1C				1C	AE	20	AE	9E 0002A	MOVAB	FLUSHBUF, PUTCACHEBUF	0891
						57	D4	0002F 1\$:	CLRL	BUCKET	0922
				10	AE	08	BA47	DE 00031 2\$:	MOVAL	@8(R10)[BUCKET], LAST_ENTRY	0913
					5B	08	BA47	D0 00037	MOVL	@8(R10)[BUCKET], ENTRY	0914
						03	12	0003C 3\$:	BNEQ	5\$	0919
						0137	31	0003E 4\$:	BRW	30\$	
					04	08	AC	E8 00041 5\$:	BLBS	WRITESTATUS, 6\$	
					F5	OC	AC	E9 00045	BLBC	CONTINUE_ON_ERR, 4\$	
				18	AE		6B	D0 00049 6\$:	MOVL	(ENTRY), NXTCACHE_ENTRY	0921
					01		01	ED 0004D	CMPZV	#1, #1, 12(ENTRY), FLUSH_DATA	0922
						03	13	00054	BEQL	7\$	
						0114	31	00056	BRW	28\$	
					03	OC	AB	E8 00059 7\$:	BLBS	12(ENTRY), 9\$	0925
						00F2	31	0005D 8\$:	BRW	27\$	
					F9	08	AC	E9 00060 9\$:	BLBC	WRITESTATUS, 8\$	0928
					AE	04	AB	D0 00064	MOVL	4(ENTRY), TARGVBN	0935
				14	BB	0200	8F	28 00069	MOVC3	#512, @8(ENTRY), @PUTCACHEBUF	0937
					58		01	D0 00071	MOVL	#1, BUFBLKCNT	0938
					OC		01	D0 00074	MOVL	#1, LSTMODBLKLOC	0939
				08	AE		01	C1 00078	ADDL3	#1, 4(ENTRY), NEXT_VBN	0940
					04		57	D1 0007E	CMPL	BUCKET, #127	0947
					8F		06	18 00085	BGEQ	10\$	
						01	A7	9E 00087	MOVAB	1(R7), NEXT_BUCKET	0948
							02	11 0008B	BRB	11\$	
							56	D4 0008D 10\$:	CLRL	NEXT_BUCKET	0949
					04	AE	58	D1 0008F 11\$:	CMPL	BUFB[KCNT, PUTBUFSIZ	0955
							03	19 00093	BLSS	13\$	
						009B	31	00095 12\$:	BRW	26\$	
						08	AE	D5 00098 13\$:	TSTL	NEXT_VBN	
							F8	13 0009B	BEQL	12\$	
					50	08	BA46	D0 0009D	MOVL	@8(R10)[NEXT_BUCKET], NXTBUCKET_ENTRY	0961
					51	08	BA46	DE 000A2	MOVAL	@8(R10)[NEXT_BUCKET], LAST_BUCKET_ENTRY	0962
							59	D4 000A7	CLRL	NXTVBN_ENTRY	0963
							50	D5 000A9 14\$:	TSTL	NXTBUCKET_ENTRY	0971
							2A	13 000AB	BEQL	17\$	
							59	D5 000AD	TSTL	NXTVBN_ENTRY	
							26	12 000AF	BNEQ	17\$	
				08	AE	04	A0	D1 000B1	CMPL	4(NXTBUCKET_ENTRY), NEXT_VBN	0973
							08	15 000B6	BLEQ	15\$	
					51		50	D0 000B8	MOVL	NXTBUCKET_ENTRY, LAST_BUCKET_ENTRY	0976
					50		60	D0 000BB	MOVL	(NXTBUCKET_ENTRY), NXTBUCKET_ENTRY	0977
							E9	11 000BE	BRB	14\$	0973
							11	12 000C0 15\$:	BNEQ	16\$	0981
					01		01	ED 000C2	CMPZV	#1, #1, 12(NXTBUCKET_ENTRY), FLUSH_DATA	0984
							08	12 000C9	BNEQ	16\$	
					61		60	D0 000CB	MOVL	(NXTBUCKET_ENTRY), (LAST_BUCKET_ENTRY)	0989
					59		50	D0 000CE	MOVL	NXTBUCKET_ENTRY, NXTVBN_ENTRY	0990
							D6	11 000D1	BRB	14\$	0983
							50	D4 000D3 16\$:	CLRL	NXTBUCKET_ENTRY	0996
							D2	11 000D5	BRB	14\$	0971
							59	D5 000D7 17\$:	TSTL	NXTVBN_ENTRY	1000
							05	12 000D9	BNEQ	19\$	
						08	AE	D4 000DB	CLRL	NEXT_VBN	1001
							AF	11 000DE 18\$:	BRB	11\$	
					0A	OC	A9	E8 000E0 19\$:	BLBS	12(NXTVBN_ENTRY), 20\$	1009
					50	01	A8	9E 000E4	MOVAB	1(R8), R0	1010

04	AE	50	D1	000E8	CMPL	R0, PUTBUFSIZ	:	
		28	13	000EC	BEQL	24\$:	
50		09	78	000EE	20\$:	ASHL	#9, BUFBKCNT, R0	1015
1C BE40	08	B9	8F	28 000F2	MOV3	#512, @8(NXTVBN_ENTRY), @PUTCACHEBUF[R0]	:	
			58	D6 000FB	INCL	BUFBKCNT	:	1016
	0C		A9	E9 000FD	BLBC	12(NXTVBN_ENTRY), 21\$:	1017
			58	D0 00101	MOVL	BUFBKCNT, LSTMODBLKLOC	:	
0000007F	8F		56	D1 00105	21\$:	CMPL	NEXT_BUCKET, #127	1022
			04	18 0010C	BGEQ	22\$:	
			56	D6 0010E	INCL	NEXT_BUCKET	:	1023
			02	11 00110	BRB	23\$:	
			56	D4 00112	22\$:	CLRL	NEXT_BUCKET	1024
		08	AE	D6 00114	23\$:	INCL	NEXT_VBN	1026
			03	11 00117	BRB	25\$:	1009
		08	AE	D4 00119	24\$:	CLRL	NEXT_VBN	1034
	51		08	A9	D0 0011C	25\$:	MOVL	8(NXTVBN_ENTRY), R1
	50	0200	8F	3C 00120	MOVZWL	#512, R0	:	1038
			0000G	30 00125	BSBW	DEALLOC_MEM	:	
	51		59	D0 00128	MOVL	NXTVBN_ENTRY, R1	:	1039
	50		0E	D0 0012B	MOVL	#14, R0	:	
			0000G	30 0012E	BSBW	DEALLOC_MEM	:	
			AB	11 00131	BRB	18\$:	0955
		0C	AE	DD 00133	26\$:	PUSHL	LSTMODBLKLOC	1042
		18	AE	DD 00136	PUSHL	TARGVBN	:	
		24	AE	DD 00139	PUSHL	PUTCACHEBUF	:	
0000V	CF		03	FB 0013C	CALLS	#3, WRITE_N_BLOCKS	:	
08	AC		50	D0 00141	MOVL	R0, WRITESTATUS	:	
	09	08	AC	E8 00145	BLBS	WRITESTATUS, 27\$:	1048
	05	0C	AC	E8 00149	BLBS	CONTINUE_ON_ERR, 27\$:	
	35		6E	E8 0014D	BLBS	BUFSTATUS, 31\$:	1052
			3F	11 00150	BRB	32\$:	1054
10	BE		6B	D0 00152	27\$:	MOVL	(ENTRY), @LAST_ENTRY	1058
	51	08	AB	D0 00156	MOVL	8(ENTRY), R1	:	1062
	50	0200	8F	3C 0015A	MOVZWL	#512, R0	:	
			0000G	30 0015F	BSBW	DEALLOC_MEM	:	
	51		5B	D0 00162	MOVL	ENTRY, R1	:	1063
	50		0E	D0 00165	MOVL	#14, R0	:	
			0000G	30 00168	BSBW	DEALLOC_MEM	:	
			04	11 0016B	BRB	29\$:	0922
10	AE		5B	D0 0016D	28\$:	MOVL	ENTRY, LAST_ENTRY	1066
	5B	18	AE	D0 00171	29\$:	MOVL	NXTCACHE_ENTRY, ENTRY	1068
			FEC4	31 00175	BRW	3\$:	0919
FEAF	57	01 0000007F	8F	F1 00178	30\$:	ACBL	#127, #1, BUCKET, 2\$	
			6E	E9 00182	BLBC	BUFSTATUS, 32\$:	1072
	50	04	09	78 00185	31\$:	ASHL	#9, PUTBUFSIZ, R0	
			51	1C AE D0 0018A	MOVL	PUTCACHEBUF, R1	:	
			0000G	30 0018E	BSBW	DEALLOC_MEM	:	
		50	08	AC D0 00191	32\$:	MOVL	WRITESTATUS, R0	1073
			04	00195	RET		:	1074

; Routine Size: 406 bytes, Routine Base: \$CODE\$ + 0000

; 349 1075 1


```
1076 1 ROUTINE test_write =
1077 1 |+++
1078 1 |
1079 1 |       Write out the largest VBN in the cache to ensure that there will
1080 1 |       be sufficient space to write the library.
1081 1 |       Return the status of the write.
1082 1 |
1083 1 |---
1084 2 BEGIN
1085 2 BIND
1086 2     context = .lbr$gl_control [lbr$l_ctxptr] : BBLOCK,
1087 2     largest_vbn = context [ctx$l_hivbn];
1088 2 LOCAL
1089 2     cache_entry : REF BBLOCK,
1090 2     status;
1091 2
1092 2 IF .context[ctx$y_vronly]
1093 2 THEN RETURN lbr$_normal
1094 2 ELSE
1095 2     BEGIN
1096 2     |
1097 2     |       If largest_vbn is 0 then no blocks were modified
1098 2     |
1099 2     |       IF .largest_vbn EQL 0 THEN RETURN lbr$_normal;
1100 2     |       perform ( lookup_cache (.largest_vbn, cache_entry) );
1101 2     |       status = write_block (.cache_entry[cache$l_address], .largest_vbn);
1102 2     |       IF .status
1103 2     |       THEN RETURN write_block (.lbr$gl_control [lbr$l_oldhdrptr], 1)
1104 2     |       ELSE RETURN lbr$_writeerr;
1105 2     END;
1106 1 END; ! ROUTINE test_write
```

OFFC 00000 TEST_WRITE:

5E		04	C2	00002	WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	1076
50	0000G	CF	D0	00005	SUBL2	#4, SP	1086
50	0E	A0	D0	0000A	MOVL	LBR\$GL_CONTROL, R0	
	04	A0	95	0000E	MOVL	14(R0), R0	
		06	19	00011	TSTB	4(R0)	1092
53	46	A0	D0	00013	BLSS	1\$	
		08	12	00017	MOVL	70(R0), R3	1099
50	00000000G	8F	D0	00019	BNEQ	2\$	
		04	00020	MOVL	#LBR\$_NORMAL, R0		
51		6E	9E	00021	RET		
50		53	D0	00024	MOVAB	CACHE_ENTRY, R1	1100
	0000G	30	00027	MOVL	R3, R0		
27		50	E9	0002A	BSBW	LOOKUP_CACHE	
52		6E	D0	0002D	BLBC	STATUS, 4\$	
51		53	D0	00030	MOVL	CACHE_ENTRY, R2	1101
50	08	A2	D0	00033	MOVL	R3, R1	
	0000G	30	00037	MOVL	8(R2), R0		
10		50	E9	0003A	BSBW	WRITE_BLOCK	
52	0000G	CF	D0	0003D	BLBC	STATUS, 3\$	1102
				MOVL	LBR\$GL_CONTROL, R2		1103

LBR_PUTCACHE
V04=000

C 3
16-Sep-1984 02:06:42
14-Sep-1984 12:37:47

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]PUTCACHE.B32;1 Page 11 (4)

51		01	D0	00042	MOVL	#1, R1	:
50	1A	A2	D0	00045	MOVL	26(R2), R0	:
		0000G	30	00049	BSBW	WRITE_BLOCK	:
			04	0004C	RET		:
50	00000000G	8F	D0	0004D 3\$:	MOVL	#LBR\$_WRITEERR, R0	1104
			04	00054 4\$:	RET		1106

; Routine Size: 85 bytes, Routine Base: \$CODE\$ + 0196

; 382 1107 1


```

: 384 1108 1 ROUTINE write_n_blocks ( addr, targetvbn, nblocks ) =
: 385 1109 2 BEGIN
: 386 1110 2
: 387 1111 2 ---
: 388 1112 2
: 389 1113 2 This routine writes out #[nblocks] blocks from the flush_cache_buffer
: 390 1114 2 in memory to the library file at the specified block number.
: 391 1115 2
: 392 1116 2 Inputs:
: 393 1117 2
: 394 1118 2 addr = address of buffer to write from
: 395 1119 2 targetvbn = first disk block number in library file to write to
: 396 1120 2 nblocks = number of blocks in buffer to write out
: 397 1121 2
: 398 1122 2 Outputs:
: 399 1123 2
: 400 1124 2 Return write status.
: 401 1125 2 ---
: 402 1126 2
: 403 1127 2 BIND
: 404 1128 2 context = .lbr$gl_control [lbr$l_ctxptr]: BBLOCK,
: 405 1129 2 rab = .context [ctx$l_recrab]: BBLOCK;
: 406 1130 2
: 407 1131 2 LOCAL
: 408 1132 2 status;
: 409 1133 2
: 410 1134 2 rab [rab$l_bkt] = .targetvbn; ! Set block number to write
: 411 1135 2 rab [rab$l_rbf] = .addr; !Set address of buffer for write
: 412 1136 2 rab [rab$w_rsz] = lbr$c_pagesize * .nblocks; !And its length
: 413 1137 2 status = $WRITE (RAB=rab); !Write the record
: 414 1138 2 IF NOT .status
: 415 1139 2 THEN
: 416 1140 3 BEGIN
: 417 1141 3 lbr$gl_rmsstv = .rab [rab$l_stv]; ! Record error code.
: 418 1142 3 RETURN lbr$_writeerr;
: 419 1143 3 END
: 420 1144 2 ELSE RETURN lbr$_normal;
: 421 1145 2
: 422 1146 1 END;
```

.EXTRN SYSS\$WRITE

0004 00000 WRITE_N_BLOCKS:

		50	0000G	CF	D0	00002	.WORD	Save R2	: 1108	
		50	0E	A0	D0	00007	MOVL	LBR\$GL_CONTROL, R0	: 1128	
		52	0C	A0	D0	0000B	MOVL	14(R0), R0		
							MOVL	12(R0), R2	: 1129	
		38	A2	08	AC	D0	0000F	MOVL	TARGETVBN, 56(R2)	: 1134
		28	A2	04	AC	D0	00014	MOVL	ADDR, 40(R2)	: 1135
22	A2	0C	AC	0200	8F	A5	00019	MULW3	#512, NBLOCKS, 34(R2)	: 1136
					52	DD	00021	PUSHL	R2	: 1137
		00000000G	00		01	FB	00023	CALLS	#1, SYSS\$WRITE	
			0E		50	E8	0002A	BLBS	STATUS, 1\$: 1138
		0000G	CF	0C	A2	D0	0002D	MOVL	12(R2), LBR\$GL_RMSSTV	: 1141
			50	00000000G	8F	D0	00033	MOVL	#LBR\$_WRITEERR, R0	: 1144

LBR_PUTCACHE
V04=000

E 3
16-Sep-1984 02:06:42
14-Sep-1984 12:37:47

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]PUTCACHE.B32;1 Page 13 (5)

50 00000000G 8F 04 0003A RET
DO 0003B 1\$: MOVL #LBR\$_NORMAL, R0
04 00042 RET

:
:
: 1146

; Routine Size: 67 bytes, Routine Base: \$CODE\$ + 01EB

; 423 1147 1


```

425 1148 1 GLOBAL ROUTINE dealloc_cache =
426 1149 2 BEGIN
427 1150 2
428 1151 2 !+++
429 1152 2
430 1153 2 Flush out the cache, writing blocks to library file and
431 1154 2 returning virtual memory. Then deallocate the hash table.
432 1155 2 First write out the block of the highest VBN to ensure sufficient disk
433 1156 2 space. Then write out data blocks then index blocks. If a write error
434 1157 2 occurs, continue deallocating the cache but quit writing the blocks out
435 1158 2 to the library file.
436 1159 2
437 1160 2 ---
438 1161 2
439 1162 2 BIND
440 1163 2 context = .lbr$gl_control [lbr$l_ctxptr] : BBLOCK;
441 1164 2
442 1165 2 LOCAL
443 1166 2 status;
444 1167 2
445 1168 2 LITERAL
446 1169 2 continue = true; ! if flush cache encounters a write error
447 1170 2 ! quit writing but continue to deallocate cache.
448 1171 2
449 1172 2 If the library has been opened read only then just empty the
450 1173 2 cache, otherwise use flush_cache which does more error checking
451 1174 2 and writes out data blocks before the index blocks.
452 1175 2
453 1176 2 IF .context[ctx$rv_only]
454 1177 2 THEN
455 1178 2 status = empty_cache ()
456 1179 2 ELSE
457 1180 2 BEGIN
458 1181 2
459 1182 2 Write out the block in the cache with the largest VBN to
460 1183 2 ensure sufficient disk space before flushing the cache.
461 1184 2
462 1185 2 status = test_write ();
463 1186 2
464 1187 2 status = flush_cache( lbr$c_flushdata, .status, continue);
465 1188 2 status = flush_cache( lbr$c_flushall, .status, continue);
466 1189 2 dealloc_mem ( [lbr$c_hashsize, .context [ctx$l_cache] );
467 1190 2 context [ctx$l_cache] = 0;
468 1191 2 END;
469 1192 2
470 1193 2
471 1194 2 If an error hasn't occurred, and cache header entry common allocation
472 1195 2 block is not empty then deallocate it.
473 1196 2
474 1197 2 IF (.status) AND (.context [ctx$l_chdallsiz] NEQ 0)
475 1198 2 THEN
476 1199 2 BEGIN
477 1200 2 dealloc_mem ( .context [ctx$l_chdallsiz], .context [ctx$l_chdalladr] );
478 1201 2 END;
479 1202 2
480 1203 2 RETURN .status;
481 1204 2 END;
```


			OFFC 00000		.ENTRY	DEALLOC CACHE, Save R2,R3,R4,R5,R6,R7,R8,-	
	50	0000G	CF D0 00002		MOVL	R9,R10,R11	1148
	52	0E	A0 D0 00007		MOVL	LBR\$GL_CONTROL, R0	1163
		04	A2 95 0000B		TSTB	14(R0), R2	
			0A 18 0000E		BGEQ	4(R2)	1176
0000G	CF		00 FB 00010		CALLS	#0, EMPTY_CACHE	1178
	53		50 D0 00015		MOVL	R0, STATUS	
			33 11 00018		BRB	2\$	
FF49	CF		00 FB 0001A 1\$:		CALLS	#0, TEST_WRITE	1185
	53		50 D0 0001F		MOVL	R0, STATUS	
			01 DD 00022		PUSHL	#1	1187
			53 DD 00024		PUSHL	STATUS	
			01 DD 00026		PUSHL	#1	
FDA5	CF		03 FB 00028		CALLS	#3, FLUSH_CACHE	
	53		50 D0 0002D		MOVL	R0, STATUS	
			01 DD 00030		PUSHL	#1	1188
			53 DD 00032		PUSHL	STATUS	
			7E D4 00034		CLRL	-(SP)	
FD97	CF		03 FB 00036		CALLS	#3, FLUSH_CACHE	
	53		50 D0 0003B		MOVL	R0, STATUS	
	51	08	A2 D0 0003E		MOVL	8(R2), R1	1189
	50	0200	8F 3C 00042		MOVZWL	#512, R0	
			0000G 30 00047		BSBW	DEALLOC_MEM	
		08	A2 D4 0004A		CLRL	8(R2)	1190
	0C		53 E9 0004D 2\$:		BLBC	STATUS, 3\$	1197
		4A	A2 D5 00050		TSTL	74(R2)	
			07 13 00053		BEQL	3\$	
	50	4A	A2 7D 00055		MOVQ	74(R2), R0	1200
			0000G 30 00059		BSBW	DEALLOC_MEM	
	50		53 D0 0005C 3\$:		MOVL	STATUS, -R0	1203
			04 0005F		RET		1204

; Routine Size: 96 bytes, Routine Base: \$CODE\$ + 022E

; 482 1205 1


```

484 1206 1 GLOBAL ROUTINE lbr$flush ( ctl_index, blktyp_to_flush ) =
485 1207 2 BEGIN
486 1208 2
487 1209 2 +++
488 1210 2
489 1211 2 FUNCTIONAL DESCRIPTION
490 1212 2
491 1213 2 This routine empties the cache of all data blocks or of all
492 1214 2 data and index blocks, and deallocates the virtual memory.
493 1215 2 If a write error is encountered then it quits both writing
494 1216 2 the cache to the library, and deallocating the cache.
495 1217 2
496 1218 2 CALLING SEQUENCE
497 1219 2
498 1220 2 status = lbr$flush ( ctl_index, blktyp_to_flush )
499 1221 2
500 1222 2 INPUT PARAMETERS
501 1223 2
502 1224 2 ctl_index:
503 1225 2 address of control table index.
504 1226 2
505 1227 2 blktyp_to_flush:
506 1228 2 If blktyp_to_flush = lbr$flushdata then write out
507 1229 2 data blocks from cache and return virtual memory.
508 1230 2 If blktyp_to_flush = lbr$flushall then write out
509 1231 2 data blocks and index blocks to library file and
510 1232 2 deallocate the virtual memory.
511 1233 2
512 1234 2 IMPLICIT OUTPUTS
513 1235 2
514 1236 2 All the blocks of the type specified are emptied from the
515 1237 2 virtual memory cache and written to the library.
516 1238 2
517 1239 2 RETURN VALUE
518 1240 2
519 1241 2 lbr$normal Success code
520 1242 2 lbr$badparam Block type other than lbr$flushall or lbr$flushdata
521 1243 2 requested.
522 1244 2 lbr$writeerr Write error during writing out of cache
523 1245 2 ---
524 1246 2
525 1247 2 LOCAL
526 1248 2 blocktype, ! flag to signal whether data blocks or both
527 1249 2 status; ! data and index blocks should be flushed.
528 1250 2
529 1251 2 LITERAL
530 1252 2 continue = false;
531 1253 2
532 1254 2 BUILTIN
533 1255 2 NULLPARAMETER;
534 1256 2
535 1257 2 perform (validate_ctl (..ctl_index));
536 1258 2 perform (check_lock() ); ! check that index is not locked
537 1259 2
538 1260 2 IF NULLPARAMETER(1)
539 1261 2 THEN blocktype = lbr$flushall
540 1262 2
```



```

: 541      1263 2      ELSE blocktype = .blktyp to flush;
: 542      1264 3      IF ( (.blocktype NEQ lbr$c_flushdata) AND (.blocktype NEQ lbr$c_flushall) )
: 543      1265 2      THEN RETURN lbr$_badparam;
: 544      1266 2
: 545      1267 2      !
: 546      1268 2      !       Write out the block in the cache with the largest VBN to
: 547      1269 2      !       ensure sufficient disk space before flushing the cache.
: 548      1270 2      !
: 549      1271 2      status = test_write ();
: 550      1272 2
: 551      1273 2      IF .status
: 552      1274 2      THEN status = flush_cache( lbr$c_flushdata, .status, continue);
: 553      1275 2      IF .blocktype EQL lbr$c_flushall
: 554      1276 2      THEN
: 555      1277 2          IF .status
: 556      1278 2          THEN status = flush_cache( lbr$c_flushall, .status, continue);
: 557      1279 2      RETURN .status;
: 558      1280 1      END;
```

		OFFC 00000		.ENTRY	LBR\$FLUSH, Save R2,R3,R4,R5,R6,R7,R8,R9,-	
					R10,R11	1206
50	04	BC D0 00002		MOVL	@CTL_INDEX, R0	1258
		0000G 30 00006		BSBW	VALIDATE_CTL	
4D		50 E9 00009		BLBC	STATUS, 5\$	
		0000G 30 0000C		BSBW	CHECK_LOCK	1259
47		50 E9 0000F		BLBC	STATUS, 6\$	
		6C 95 00012		TSTB	(AP)	1261
		05 13 00014		BEQL	1\$	
	04	AC D5 00016		TSTL	4(AP)	
		04 12 00019		BNEQ	2\$	
		52 D4 0001B 1\$:		CLRL	BLOCKTYPE	1262
		04 11 0001D		BRB	3\$	
52	08	AC D0 0001F 2\$:		MOVL	BLKTYP TO_FLUSH, BLOCKTYPE	1263
01		52 D1 00023 3\$:		CMPL	BLOCKTYPE, #1	1264
		0C 13 00026		BEQL	4\$	
		52 D5 00028		TSTL	BLOCKTYPE	
		08 13 0002A		BEQL	4\$	
50	00000000G	8F D0 0002C		MOVL	#LBR\$_BADPARAM, R0	1265
		04 00033		RET		
FECF	CF	00 FB 00034 4\$:		CALLS	#0, TEST_WRITE	1271
	0B	50 E9 00039		BLBC	STATUS, 5\$	1273
		7E D4 0003C		CLRL	-(SP)	1274
		50 DD 0003E		PUSHL	STATUS	
		01 DD 00040		PUSHL	#1	
FD2B	CF	03 FB 00042		CALLS	#3, FLUSH_CACHE	
		52 D5 00047 5\$:		TSTL	BLOCKTYPE	1275
		0E 12 00049		BNEQ	6\$	
	0B	50 E9 0004B		BLBC	STATUS, 6\$	1277
		7E D4 0004E		CLRL	-(SP)	1278
		50 DD 00050		PUSHL	STATUS	
		7E D4 00052		CLRL	-(SP)	
FD19	CF	03 FB 00054		CALLS	#3, FLUSH_CACHE	
		04 00059 6\$:		RET		1280

LBR_PUTCACHE
V04=000

J 3
16-Sep-1984 02:06:42
14-Sep-1984 12:37:47

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]PUTCACHE.B32;1 Page 18
(7)

; Routine Size: 90 bytes, Routine Base: \$CODE\$ + 028E

; 559 1281 1
; 560 1282 1 END ! module PUTCACHE
; 561 1283 0 ELUDOM

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	744	NOVEC,NOWRT, RD , EXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	21	0	581	00:01.0

COMMAND QUALIFIERS

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LISS:PUTCACHE/OBJ=OBJ\$:PUTCACHE MSRC\$:PUTCACHE/UPDATE=(ENH\$:PUTCACHE)

; Size: 744 code + 0 data bytes
; Run Time: 00:21.3
; Elapsed Time: 00:46.8
; Lines/CPU Min: 3612
; Lexemes/CPU-Min: 25869
; Memory Used: 203 pages
; Compilation Complete

0200 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

TRANSFER
LIS

DATABASE
LIS

PUTCACHE
LIS

LIBRAR

PREFIX
REQ

LIBRARIAN
MAP

CROSS
LIS

SUBS
LIS

PADLBR
LIS

COMPRESS
LIS

LIB
MDL

FILEIO
LIS

EXTRACT
LIS

DELETE
LIS